```
import serial
import time
from time import process_time
import numpy as np
import pyqtgraph as pg
from pyqtgraph.Qt import QtGui
from google_speech import Speech
import math

# Change the configuration file name
configFileName = 'xwr16xx_7677.cfg'

CLIport = {}
Dataport = {}
byteBuffer = np.zeros(2**15,dtype = 'uint8')
byteBufferLength = 0;



# ------------------------------------------------------------------

# Function to configure the serial ports and send the data from
# the configuration file to the radar
def serialConfig(configFileName):

    global CLIport
    global Dataport
    # Open the serial ports for the configuration and the data ports

    # Raspberry pi
    CLIport = serial.Serial('/dev/ttyACM0', 115200)
    Dataport = serial.Serial('/dev/ttyACM1', 921600)

    # Windows
    #CLIport = serial.Serial('COM8', 115200)
    #Dataport = serial.Serial('COM9', 921600)

    # Read the configuration file and send it to the board
    config = [line.rstrip('\r\n') for line in open(configFileName)]
    for i in config:
        CLIport.write((i+'\n').encode())
        print(i)
        time.sleep(0.01)

    return CLIport, Dataport

# ------------------------------------------------------------------

# Function to parse the data inside the configuration file
def parseConfigFile(configFileName):
    configParameters = {} # Initialize an empty dictionary to store the
configuration parameters

    # Read the configuration file and send it to the board
    config = [line.rstrip('\r\n') for line in open(configFileName)]
    for i in config:

        # Split the line
        splitWords = i.split(" ")

        # Hard code the number of antennas, change if other configuration is used
        numRxAnt = 4
        numTxAnt = 3

        # Get the information about the profile configuration
        if "profileCfg" in splitWords[0]:
            startFreq = int(float(splitWords[2]))
            idleTime = int(splitWords[3])
            rampEndTime = float(splitWords[5])
            freqSlopeConst = float(splitWords[8])
            numAdcSamples = int(splitWords[10])
            numAdcSamplesRoundTo2 = 1;

            while numAdcSamples > numAdcSamplesRoundTo2:
                numAdcSamplesRoundTo2 = numAdcSamplesRoundTo2 * 2;

            digOutSampleRate = int(splitWords[11]);

        # Get the information about the frame configuration
        elif "frameCfg" in splitWords[0]:

            chirpStartIdx = int(splitWords[1]);
            chirpEndIdx = int(splitWords[2]);
            numLoops = int(splitWords[3]);
            numFrames = int(splitWords[4]);
            framePeriodicity = float(splitWords[5]);


    # Combine the read data to obtain the configuration parameters
    numChirpsPerFrame = (chirpEndIdx - chirpStartIdx + 1) * numLoops
    configParameters["numDopplerBins"] = numChirpsPerFrame / numTxAnt
    configParameters["numRangeBins"] = numAdcSamplesRoundTo2
    configParameters["rangeResolutionMeters"] = (3e8 * digOutSampleRate * 1e3) /
(2 * freqSlopeConst * 1e12 * numAdcSamples)
    configParameters["rangeIdxToMeters"] = (3e8 * digOutSampleRate * 1e3) / (2 *
freqSlopeConst * 1e12 * configParameters["numRangeBins"])
    configParameters["dopplerResolutionMps"] = 3e8 / (2 * startFreq * 1e9 *
(idleTime + rampEndTime) * 1e-6 * configParameters["numDopplerBins"] * numTxAnt)
    configParameters["maxRange"] = (300 * 0.9 * digOutSampleRate)/(2 *
freqSlopeConst * 1e3)
    configParameters["maxVelocity"] = 3e8 / (4 * startFreq * 1e9 * (idleTime +
rampEndTime) * 1e-6 * numTxAnt)

    return configParameters

# ------------------------------------------------------------------
def compare_range(r,a):
    dist_string = {True: "Object at less than 20 meters",False:{True: "Object at
less than ten meters",False:{True: "Object at less than five
meters",False:"Danger"}[math.floor(r)<=5]}[math.floor(r)<=10 and math.floor(r)>5]}
[math.floor(r)<=20 and math.floor(r)>10]
    direc_string = {True: " at left", False:" at right"}[a<0]
    return (dist_string + direc_string)
# Funtion to read and parse the incoming data
def readAndParseData18xx(Dataport, configParameters):
    global byteBuffer, byteBufferLength

    # Constants
    OBJ_STRUCT_SIZE_BYTES = 12;
    BYTE_VEC_ACC_MAX_SIZE = 2**15;
    MMWDEMO_UART_MSG_DETECTED_POINTS = 1;
    MMWDEMO_UART_MSG_RANGE_PROFILE   = 2;
    maxBufferSize = 2**15;
    tlvHeaderLengthInBytes = 8;
    pointLengthInBytes = 16;
    magicWord = [2, 1, 4, 3, 6, 5, 8, 7]

    # Initialize variables
    magicOK = 0 # Checks if magic number has been read
    dataOK = 0 # Checks if the data has been read correctly
    frameNumber = 0
    detObj = {}
    readBuffer = Dataport.read(Dataport.in_waiting)
    byteVec = np.frombuffer(readBuffer, dtype = 'uint8')
    byteCount = len(byteVec)

    # Check that the buffer is not full, and then add the data to the buffer
    if (byteBufferLength + byteCount) < maxBufferSize:
        byteBuffer[byteBufferLength:byteBufferLength + byteCount] = 
byteVec[:byteCount]
        byteBufferLength = byteBufferLength + byteCount

    # Check that the buffer has some dataimport numpy as np
    if byteBufferLength > 16:

        # Check for all possible locations of the magic word
        possibleLocs = np.where(byteBuffer == magicWord[0])[0]

        # Confirm that is the beginning of the magic word and store the index in 
startIdx
        startIdx = []
        for loc in possibleLocs:
            check = byteBuffer[loc:loc+8]
            if np.all(check == magicWord):
                startIdx.append(loc)

        # Check that startIdx is not empty
        if startIdx:

            # Remove the data before the first start index
            if startIdx[0] > 0 and startIdx[0] < byteBufferLength:
                byteBuffer[:byteBufferLength-startIdx[0]] = 
byteBuffer[startIdx[0]:byteBufferLength]
                byteBuffer[byteBufferLength-startIdx[0]:] = 
np.zeros(len(byteBuffer[byteBufferLength-startIdx[0]:]),dtype = 'uint8')
                byteBufferLength = byteBufferLength - startIdx[0]

            # Check that there have no errors with the byte buffer length
            if byteBufferLength < 0:
                byteBufferLength = 0

            # word array to convert 4 bytes to a 32 bit number
            word = [1, 2**8, 2**16, 2**24]

            # Read the total packet length
            totalPacketLen = np.matmul(byteBuffer[12:12+4],word)

            # Check that all the packet has been read
            if (byteBufferLength >= totalPacketLen) and (byteBufferLength != 0):
                magicOK = 1

    # If magicOK is equal to 1 then process the message
    if magicOK:
        # word array to convert 4 bytes to a 32 bit number
        word = [1, 2**8, 2**16, 2**24]

        # Initialize the pointer index
        idX = 0

        # Read the header
        magicNumber = byteBuffer[idX:idX+8]
        idX += 8
        version = format(np.matmul(byteBuffer[idX:idX+4],word),'x')
        idX += 4
        totalPacketLen = np.matmul(byteBuffer[idX:idX+4],word)
        idX += 4
        platform = format(np.matmul(byteBuffer[idX:idX+4],word),'x')
        idX += 4
        frameNumber = np.matmul(byteBuffer[idX:idX+4],word)
        idX += 4
        timeCpuCycles = np.matmul(byteBuffer[idX:idX+4],word)
        idX += 4
        numDetectedObj = np.matmul(byteBuffer[idX:idX+4],word)
        idX += 4
        numTLVs = np.matmul(byteBuffer[idX:idX+4],word)
        idX += 4
        subFrameNumber = np.matmul(byteBuffer[idX:idX+4],word)
        idX += 4

        # Read the TLV messages
        for tlvIdx in range(numTLVs):

            # word array to convert 4 bytes to a 32 bit number
            word = [1, 2**8, 2**16, 2**24]

            # Check the header of the TLV message
            tlv_type = np.matmul(byteBuffer[idX:idX+4],word)
            idX += 4
            tlv_length = np.matmul(byteBuffer[idX:idX+4],word)
            idX += 4

            # Read the data depending on the TLV message
            if tlv_type == MMWDEMO_UART_MSG_DETECTED_POINTS:

                # Initialize the arrays
                x = np.zeros(numDetectedObj,dtype=np.float64)
                y = np.zeros(numDetectedObj,dtype=np.float64)
                z = np.zeros(numDetectedObj,dtype=np.float64)
                velocity = np.zeros(numDetectedObj,dtype=np.float64)
                range_obj = np.zeros(numDetectedObj,dtype=np.float64)

                print("Number of objects Detected: ", numDetectedObj)
                for objectNum in range(numDetectedObj):

                    # Read the data for each object
                    x[objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)
                    idX += 4
                    y[objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)
                    idX += 4
                    z[objectNum] = byteBuffer[idX:idX + 4].view(dtype=np.float32)
                    idX += 4
                    point1 = np.array((0, 0, 0))
                    point2 = np.array((x[objectNum], y[objectNum], z[objectNum]))
                    dist = np.linalg.norm(point1 - point2)
                    range_obj[objectNum] = dist;
                    velocity[objectNum] = byteBuffer[idX:idX +
4].view(dtype=np.float32)

                    if( x[objectNum]< 0):
                        print("Position of [",objectNum,"]: Left")
                    if (x[objectNum] > 0):
                        print("Position of [",objectNum,"]: Right")
                    print("Velocity[", objectNum,"]: ",velocity[objectNum])
                    print("Range[", objectNum,"]:  ",range_obj[objectNum])
                    idX += 4
                # Store the data in the detObj dictionary
                detObj = {"numObj": numDetectedObj, "x": x, "y": y, "z": z,
"velocity":velocity,"Range":range_obj}
                dataOK = 1
                angle_obj = np.arctan(detObj["x"]/detObj["y"])*180/3.14
                print("Angle",angle_obj, end='\n')
                #engine = pyttsx3.init()
                #engine.setProperty('rate', 150)
                '''
                sox_effects = ("speed", "1.25")
                lang = "en"
                for r,a in zip(detObj["Range"],angle_obj):
                    if(compare_range(r,a)!="Danger at right" or compare_range(r,a)!
="Danger at left"):
                        #engine.say(compare_range(r))
                        print(compare_range(r,a))
                        text = compare_range(r,a)
                        speech = Speech(text, lang)
                        speech.play(sox_effects)
                        #engine.runAndWait()
                #engine.stop()
                '''

        # Remove already processed data
        if idX > 0 and byteBufferLength>idX:
            shiftSize = totalPacketLen


            byteBuffer[:byteBufferLength - shiftSize] = 
byteBuffer[shiftSize:byteBufferLength]
            byteBuffer[byteBufferLength - shiftSize:] = 
np.zeros(len(byteBuffer[byteBufferLength - shiftSize:]),dtype = 'uint8')
            byteBufferLength = byteBufferLength - shiftSize

            # Check that there are no errors with the buffer length
            if byteBufferLength < 0:
                byteBufferLength = 0

    return dataOK, frameNumber, detObj

# ------------------------------------------------------------------

# Funtion to update the data and display in the plot
def update():
    dataOk = 0
    global detObj
    x = []
    y = []
    vel=[]
    # Read and parse the received data
    dataOk, frameNumber, detObj = readAndParseData18xx(Dataport, configParameters)

    if dataOk and len(detObj["x"])>0:
        #print(detObj)
        x = -detObj["x"]
        y = detObj["y"]
        #vel= detObj["velocity"]

        s.setData(x,y)
        QtGui.QApplication.processEvents()

    return dataOk


# -------------------------    MAIN   -----------------------------------------

# Configurate the serial port
CLIport, Dataport = serialConfig(configFileName)

# Get the configuration parameters from the configuration file
configParameters = parseConfigFile(configFileName)

# START QtAPPfor the plot
app = QtGui.QApplication([])
```

```
# Set the plot
pg.setConfigOption('background','w')
win = pg.GraphicsWindow(title="2D scatter plot")
p = win.addPlot()
p.setXRange(-10.0,10.0)
p.setYRange(0,20)

p.setLabel('left',text = 'Y position (m)')
p.setLabel('bottom', text= 'X position (m)')
s = p.plot([],[],pen=None,symbol="o")


# Main loop
detObj = {}
frameData = {}
currentIndex = 0

while True:
    t1_start = process_time()
    try:
        # Update the data and check if the data is okay
        dataOk = update()

        if dataOk:
            # Store the current frame into frameData
            frameData[currentIndex] = detObj
            currentIndex += 1

        time.sleep(0.05) # Sampling frequency of 30 Hz
        t1_stop = process_time()
        print("Elapsed time in seconds:", t1_stop - t1_start)

    # Stop the program and close everything if Ctrl + c is pressed
    except KeyboardInterrupt:
        CLIport.write(('sensorStop\n').encode())
        CLIport.close()
        Dataport.close()
        win.close()
        break
```